1 Usage

1.1 The problem

Often package writers want to redefine certain macros to do slightly more than what they did previously, adding a control sequence or two to the beginning or the end of the definition. The easiest way to accomplish this is to use something like

\let\old@macro\macro
\def\macro{...\old@macro...}

But this sort of construction can cause problems if another package also wants to redefine the same macro and happens to choose the same name to save it to. It’s also an ugly solution in that it pollutes the global namespace with extra macro names. A much cleaner solution is to define the new macro with the old macro expanded inline, as in \edef\macro{...\macro...}. This is generally problematic because there are often undefined control sequences and macros that we don’t want to expand quite yet. A compromise is to use \expandafter, but this leads to error-prone and unreadable code:

\expandafter\def\expandafter\macro\expandafter{\expandafter...
\expandafter\macro...

1.2 The solution

What we really want is a way to expand just a few tokens in the definition and leave the rest untouched. We provide a command \Inline that can be inserted before a \def or \gdef (optionally prefixed by \long, \outer, and/or \global, as in \Inline\long\outer\gdef...). Within \Inline definitions, only tokens preceded by \Expand are expanded. Thus, the previous example becomes

\Inline\def\macro{...\Expand\macro...}

\*
email: sdh33@cornell.edu
1.3 Special commands

While nearly everything can be done with \Expand alone, we provide a few more keywords for completeness and convenience.

\Expand

- \Expand - Performs a single expansion on the token or group immediately following and places the result directly into the definition without further processing. In the case of a group, only the first token is expanded (although \expandafter may be used to expand a different token), and the outermost grouping braces are discarded.

\MultiExpand{(number)}

- \MultiExpand{(number)} - Expands the following token or group the given number of times. For example,

```
\MultiExpand{3}\{\expandafter\expandafter\expandafter\a
\expandafter\b\c\}
```

expands first \c, then \b, then \a, and inserts the whole expansion into the definition with no braces. Note that the braces are important. Otherwise it will just try to expand the first \expandafter three times, which is clearly wrong.

\UnsafeExpand

- \UnsafeExpand - This version simply inserts an \expandafter, performing the expansion as in \Expand above, but reinserting the result back into the stream to be processed. Thus, any tokens like \Expand or \Super in the expansion will be acted on. Unlike the previous two commands, groups are not treated differently.

\NoExpand

- \NoExpand - If a token is preceeded by \NoExpand then it is inserted in the definition exactly as-is. This is required to insert any of the special tokens \Expand, \NoExpand, etc, as well as the internal token \Q@END, into a definition. If the token immediately following \NoExpand is an open-brace then the entire text of the group will be inserted without expansion, and the outer level of grouping will be lost.

\Super

- \Super - When redefining an already existing macro, \Super will expand to the previous definition of the macro. Any macro parameters are automatically substituted. If the macro is undefined, or if the new parameter text doesn’t match with the old text, then this will cause an error.

\Recurse

- \Recurse - This is complementary to \Super and, while not strictly necessary, is included for clarity. \Recurse is equivalent to \NoExpand\macro when defining \macro. However, since \macro is, by default, not expanded anyway, this is a bit redundant.
1.4 Calling options

When the name of the macro we’re defining is encountered, there are three different ways we might proceed: leave it alone (\NoExpand), expand it with implicit parameters (\Super), or expand it with explicit parameters (\UnsafeExpand). We therefore allow zero, one, or two stars to come after \Inline to change this behavior.

- \Inline - Without any stars, we default to leaving the macro name alone, as in \NoExpand\macro. This is the most consistent behavior with the rest of the package and works regardless of whether the macro is being defined or redefined.

- \Inline* - With a single star, we treat the macro name as a call to \Super and expand it with parameters inserted automatically. This is preferred over \Expand because it doesn’t lead to the possible surprises in the case of recursively-defined macros.

- \Inline** - Finally, with two stars, the macro name is treated as if it were preceeded by \UnsafeExpand. Any parameters must be inserted explicitly, and the expansion is itself subject to inline processing. Note that this form is the most dangerous.

One final option applies only in the case of redefining an already-existing macro. In this case, if the parameter text of the new definition differs from the parameter text of the old definition, we will produce an error. This error can be suppressed by adding a bang to the end of \Inline (either before or after the stars), acknowledging that any ill consequences that result are your own fault.

1.5 Known issues

- If a macro is defined with a character other than # catcoded to 6, then \Super will fail unless the same character is used in the redefinition.

1.6 Related packages

moredefs The moredefs package in the frankenstein collection provides some similar syntactic sugar, but is not as expressive.
2 Implementation

Make the @-sign into a letter for use in macro names. As long as the packages are well-behaved, we can put this here and not later. We also define \xa to be `\expandafter` for convenience.

```latex
\let\expandafter\ifID@aborted
\ifID@star\let\expandafter\ifID@starstar
\let\expandafter\ifID@bang

We define a conditional so that we can gracefully abort in case of an error.

```latex
\newif\ifID@aborted
\newif\ifID@star
\newif\ifID@starstar
\newif\ifID@bang
```

At some point we need to stop using the internal toks registers and allocate our own, because somebody might want to `\Expand{the\toks@}` and expect something else. We can get by with a single one though by defining a `\ID@pdef` that doubles all the # signs and then does a regular def, so that `\ID@pdef\cs\ID@toks ... \cs` will be the same as `\the\ID@toks`.

```latex
\newtoks\ID@toks
\newcount\ID@count
```

These are the macros that get it all started. `\Inline` opens up a group (which is closed at the end of `\ID@def`) and initializes a toks register (we don’t bother allocating it since we’re in a group and don’t call any LaTeX or TeX macros that make use of allocated toks registers. Then we scan the tokens until we find either an `\edef` or an `\xdef`. If it’s anything else, we just add it to the toks register. We also have a list of bad tokens that will cause an error message, so that we don’t go too far before figuring out what went wrong. Should `\Inline` be `\outer`?

```latex
\DeclareRobustCommand\Inline{%
\begingroup
% Define a few ‘quarks’
\def\Expand{\Expand}\def\Super{\Super}%
\def\UnsafeExpand{\UnsafeExpand}\def\MultiExpand{\MultiExpand}%
\def\Recurse{\Recurse}\def\NoExpand{\NoExpand}%
\def\Q@END{\Q@END}%
% Define a toks register
\ID@toks{}%
% Signal that we need to look for a star
\@testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
% Start scanning for \def or \gdef
\futurelet\@foo\ID@scandef
}%
\newcommand\ID@scandef{%
% Default behavior
\let\next\ID@saveprefix
% If this is the first few tokens after the \Inline, check for * or !
\if\testtrue\ID@starfalse\ID@starstarfalse\ID@bangfalse
\futurelet\&foo\ID@scandef%
\let\next\ID@saveprefix % Default behavior
```
These just get the scandef loop started again and set \@testtrue if we’re still looking for stars and/or bangs.
\def\ID@scandef{\futurelet\@foo\ID@start\edef\@testtrue{\futurelet\@foo\ID@scandef}}
\ID@start These are the three macros called by \ID@scandef to either save the prefix (\long, \outer, etc) to a token register, (attempt to) abort the procedure in case of an error, or else get the definition started once we find the \edef or \xdef.
\ID@sd@checkagain\ID@sd@lastcheck
\ID@saveprefix\ID@abort\ID@start
% Now look for a \def or \gdef
\ifx\@foo\def\def\next{\ID@start\def}\fi
\ifx\@foo\gdef\def\next{\ID@start\gdef}\fi
% Error checking (minimal)
\@testfalse
\ifx\@foo\edef\@testtrue{\futurelet\@foo\ID@scandef}\fi
\ifx\@foo\xdef\@testtrue\fi
\ifx\@foo\newcommand\@testtrue\fi
\ifx\@foo\renewcommand\@testtrue\fi
\ifx\@foo\DeclareRobustCommand\@testtrue\fi
\if@test\PackageError{inlinedef}{Only \protect\def\space and \protect\gdef\space are allowed after \protect\Inline,\MessageBreak but some other type of definition was found}\@eha\let\next\ID@abort\fi
\@testfalse
\ifx\@foo\bgroup\@testtrue\fi\ifx\@foo\let\@testtrue\fi
\ifx\@foo\begingroup\PackageError\@testtrue\{\PackageError{inlinedef}{No \protect\def\space or \protect\gdef\space found after \protect\Inline}\MessageBreak\def\next{\ID@abort\{}}\fi
\next
\ifx\@foo*\if\ID@star
\if\ID@bang\let\next\ID@sd@lastcheck\else\let\next\ID@sd@checkagain\fi
\ID@startrue
\else
\let\next\ID@sd@checkagain
\ID@startrue
\fi
\fi
\ifx\@foo!\if\ID@bang\else % two bangs - can this be anything but an error?
\ID@bangtrue
\xa\let\xa\next\if\ID@starstar\ID@sd@lastcheck\else\ID@sd@checkagain\fi
\fi
\fi
\fi
\fi
% Now look for a \def or \gdef
\ifx\@foo\def\def\next{\ID@start\def}\fi
\ifx\@foo\gdef\def\next{\ID@start\gdef}\fi
\if\cat\noexpand\@foo\space
\def\next{\ID@toks\xa\xa\xa\{\xa\the\xa\ID@toks\space}\futurelet\xa\@foo\ID@scandef\ID@unspace}\% copied from ID@space
\fi
\fi
\PackageError{inlinedef}{Only \protect\def\space and \protect\gdef\space are allowed after \protect\Inline,\MessageBreak but some other type of definition was found}\@eha\let\next\ID@abort\fi
\@testfalse
\ifx\@foo\edef\@testtrue\fi\ifx\@foo\xdef\@testtrue\fi
\ifx\@foo\newcommand\@testtrue\fi
\ifx\@foo\renewcommand\@testtrue\fi
\ifx\@foo\DeclareRobustCommand\@testtrue\fi
\if@test\PackageError{inlinedef}{No \protect\def\space or \protect\gdef\space found after \protect\Inline}\MessageBreak\def\next{\ID@abort\{}}\fi
\next
\ifx\@foo*
In case the error was just the wrong type of \def, we consume up to and including the first explicit group.

To get the definition process started, we take \#1 as the definition command to save (either \def or \gdef), \#2 as the command that was provided (which we discard), \#3 is the name of the macro to define, and \#4 is the parameter text, delimited by a begin-group character.

In order for \Super to work properly, we need to fix the parameter list to put the \#1 in braces, since it actually consists of two tokens. Therefore, \ID@fixparams takes everything between it and \Q@END and puts it in \toks@fixedparams. If it finds a #, then it checks whether the argument is delimited or not, and if not, it inserts a pair of braces. We currently define these with \newcommand*, though if there were a reason we could conceivably make them \long. Update: we now use \ID@toks and then define \ID@fixedparams from there.

These are the two commands that \ID@fixparams calls to actually consume each token, depending on whether it was a parameter.
\futurelet\@foo\ID@fp@start
\newcommand*\ID@fp@end[1]{{%
  \xdef\xa\def\xa\ID@fixedparams\xa{\the\ID@toks}%
  \}
}\newcommand*\ID@def[3]{{%
% Other definitions
\global\ID@abortedfalse
\let\@reservedc#1%
\def\@macroname{#1}% for error message
\ID@fixparams#2\Q@END
% These are used by \Super but easier to define here
\def\@reservedb#2{}%
\edef\@reservedb{\xa\ID@getprefix\meaning\@reservedb\Q@END}%
% These are used by \Super but easier to define here
\ifx#1\undefined % hopefully nobody's going around defining \undefined
  \let\@reserveda\undefined
\else
  \edef\@reserveda{\xa\ID@getprefix\meaning#1\Q@END}%
\fi
\ifID@bang\else\ID@checkusage\fi
\ifID@aborted\else
  \ID@toks{}\ID@scan#3\Q@END{}% we need the {} so that the the #1# works...
\fi
\ifID@aborted
  \def\command{}% gracefully ignore
\else
  \let#1\relax % don’t want it expanded in the |\edef| below
% We don’t need to worry about scope anymore
  \edef\xa{\xa\ID@prefix}% (easiest way to avoid expansion...)
% % %
  \edef\command{\the\toks1\#1\the\toks2{\the\toks0}}%
% We could also write this with 3 levels of \xa...
\fi
\ID@def Here is where the “main loop” is initiated. We start by pretending to allocate another token register (though we actually just \toksdef it), and then define a number of quarks which we use as delimiters for various purposes. Finally, we start scanning. Afterwards, we test if there was an error and if not, we expand the definition command after the \endgroup so that we can clean up all the local variables.

This should deal with everything except a single #, but that’s a hairy situation in the first place and we really don’t want to allow using \Super in that case. We could probably make an error message to say so. The only other alternative would be to “go back in time” and change the last {#1} to a #1{} and even then we end up with an extra {} on the input stream. I can’t actually figure out how to test if this has happened in ...@insertp, anyway.
This is the main loop. We look at each token in turn and deal with it, mostly by inserting it into \ID@toks. If it’s a \Q@END then we’re done. If it’s \Super or \Expand then we need to do something special. If it’s a space, then we need to add the space to \ID@toks. Finally, if it’s a \bgroup then we need to figure out whether it’s an explicit or an implicit group. In the former case, we descend into it (writing \ldots to \ID@toks) and in the latter, we just pick it up like normal.

\newcommand\ID@scan{\futurelet\@foo\ID@switch}
\newcommand\ID@switch{\let
ext\ID@normal
\ifx\@foo\Q@END
\let
ext\@gobble
\fi
\ifx\@foo\@reservedc % macro name... what to do?
\ifID@star
\ifID@starstar
\let
ext\ID@expandmacro
\else
\let
ext\ID@expandsuper
\fi
\fi
\ifx\@foo\Super
\let
ext\ID@expandsuper
\fi
\ifx\@foo\Expand
\let
ext\ID@expandnext
\fi
\ifx\@foo\UnsafeExpand
\let
ext\ID@expandunsafe
\fi
\ifx\@foo\MultiExpand
\let
ext\ID@expandmulti
\fi
\ifx\@foo\NoExpand
\let
ext\ID@noexpans super
\fi
\ifcat\noexpand\@foo\space
\let
ext\ID@space
\fi
\ifcat\noexpand\@foo\bgroup
\let
ext\ID@trygroup
\fi
\def\next{\xa\xa\xa\ID@scan\xa\xa\xa\NoExpand\xa\xa\@macroname\@gobble}%
\fi
\ifcat\noexpand\@foo\space
\let
next\ID@space
\fi
\ifcat\noexpand\@foo\bgroup
\let
next\ID@trygroup
\fi
\fi
It’s a bit tricky to deal with spaces properly. In particular, picking up just a space from the token list takes some doing. We need a fully-expandable macro so that the whole thing disappears. `\ID@space` then adds a space to `\ID@toks` and then expands `\ID@unspace` after `\ID@scan` so that the `\futurelet` sees the next token after the space and can deal with it properly. We need the `\expandafter` in defining `\ID@unspace` to actually get the space token into the parameter text; otherwise, it gets gobbled up by the lexer after reading the control sequence name.

```\newcommand{\ID@space}{{% \ID@toks\xa\xa\xa\{\xa\the\ID@toks\space}% \xa\ID@scan\ID@unspace% \expandafter% \endgroup% \ID@toks\xa\xa\xa\{\xa\the\ID@toks\xa\xa\xa\{\xa\the\ID@toks}}%```

The next two macros are used to check if the `\bgroup` token was an explicit or an implicit grouping character. If it’s explicit then the next macro that takes an argument will scoop the whole thing up at once, and so we need to be aware of this to deal with it. `\ID@trygroup` uses the special `#` delimiter and compares the argument with `@empty` to see if anything comes before the next `{`. If it doesn’t find anything then it was an explicit group and we recurse. One consequence of this is that we always need to put a `{` after `\Q@END` so that we don’t get an error here.

```\newcommand{\ID@trygroup}{}\long\def{\ID@trygroup#1#}{% check for explicit/implicit grouping! % def\reservedd(#1)% % xa\let\xa\next % if\reservedd@empty\ID@recurse\else\ID@normal\fi % next#1%} %```

Here we need to do some gymnastics to get the `{` and `}` tokens into the toks register. It would be easiest if we could just add them one at a time, but we can only add balanced text, so we need to expand the whole thing first and then add it back to the register we expanded it into. Thus, we enter a new level of grouping to save the contents of `\ID@toks`, expand the inner group, and use `\expandafter` across an `\endgroup` to get the correct tokens in the right place in `\ID@toks`.

```\newcommand{\ID@recurse[]}{{% % begingroup\ID@toks{}% start a new level of grouping and empty \ID@toks % \ID@scan{}% % parse... % \xa\endgroup\xa % this fiasco should get the job done...! % \ID@toks\xa\xa{\xa\the\xa\ID@toks\xa{\the\ID@toks}}% % \ID@scan%} %```

This is what we do when it’s not anything special.
Here we define tests that will issue errors if the parameter texts aren’t the same, or the original function isn’t defined.

These correspond to the two special tokens, \Super and \Expand. The first one tests that the parameter list is alright and that the original command wasn’t undefined. If all is well, it expands everything in the right order. The second one is simpler, just inserting an \expandafter before the continuation (\ID@scan) to expand whatever comes next once. There is (yet) no way to fully-expand, although
several \Expand and \expandafter can be stacked cleverly to expand several things in a specific order.

\newcommand*{\ID@expandsuper}[1]{{% 
  \ID@checkusage\ID@checkredef 
  \if\ID@aborted\else 
    {\the\ID@toks} \@reservedc\ID@fixedparams}% 
  \fi 
  \ID@scan 
} 
\newcommand{\ID@expandnext}[2]{{% 
  \the\ID@toks \ID@scan 
} 
\newcommand{\ID@expandmulti}[3]{{% 
  \begingroup % #1 is the \MultiExpand... 
  \ID@count#2\relax % this will need to be allocated too! 
  %
  \@testtrue \ifnum\ID@count<1 \@testfalse \fi 
  \@whilesw \if@test \fi{{% 
    \the\ID@toks \ID@scan 
    \advance\ID@count\m@ne \ifnum\ID@count<1 \@testfalse \fi 
  }% 
  }% 
  \the\ID@toks \ID@scan 
} 
\newcommand*{\ID@expandunsafe}[1]{{% 
  \expandafter\ID@scan 
} 
\newcommand*{\ID@expandmacro}[1]{{% 
  \expandafter\ID@scan \@reservedc 
} 
\ID@getprefix This is used to compare argument lists. 
\newcommand\ID@getprefix{}{% 
  \long\def\ID@getprefix#1:#2->#3\Q@END{\detokenize{#2}} 
} 

Finally we clean up by restoring @’s catcode. 
\makeatother 
\end
3 Test suite

We include a somewhat-comprehensive test suite to make sure that everything is working. If it works properly, it should output nothing.

First we define a few helper-functions to test for errors, etc.

Here we predefine copies of the errors so that we can look for them easily
Now we start the actual tests.

% I. Basic stuff
% A. Simple definition
\let\a\undefined
\Inline\def\a{b}
\CheckDefinition\a{b}

% B. Simple redefinition
\def\a{b}
\Inline\def\a#1{c}
\CheckError
\CheckDefinition\a{b} % shouldn’t have changed

% C. Erroneous redefinition (needs !)
\def\a{b}
\WantNoMatchBang\a{\pound1}
\Inline\def\a#1{c}
\CheckError
\CheckDefinition\a{b} % shouldn’t have changed
% D. Local/global definition
\def\a{b}
\begingroup
\Inline\def\a{c}
\endgroup
\CheckDefinition\a{b}
\begingroup
\Inline\gdef\a{c}
\endgroup
\CheckDefinition\a{c}
{\Inline\global\def\a{d}}
\CheckDefinition\a{d}
%
% E. Collecting arguments
\Inline\long\def\a{e}
\CheckDefinition[\long]\a{e}
\Inline\outer\def\a{f}
\edef\a{\meaning\a}
\edef\b{\detokenize{\outer\macro:->f}}
\xa\CheckDefinition\xa\a\xa{\b}
\Inline\long\outer\def\a{g}
\edef\a{\meaning\a}
\edef\b{\string\long\string\outer\space\detokenize{macro:->g}}
\xa\CheckDefinition\xa\a\xa{\b}
\def\a{g}
\Inline!\long\def\a\#1{h}
\CheckDefinition[\long]\a\#1{h}
%
% II. Special tokens
%
% A. Recursion
\def\a{b}
\Inline\def\a{a\a c}
\CheckDefinition\a{a\a c}
%
% B. Expansion
\def\a{b}
\Inline\def\a{a\Expand\a c}
\CheckDefinition\a{abc}
\def\a{b}
\Inline\def\a{\Expand\a\Expand\Expand\a c}
\CheckDefinition\a{abc}
\def\b{c}
\def\b{c}
\def\a{a\text{\Expand}a c}
\CheckDefinition\a{a\ b c}
\toks0{b}\toks1{d}
\def\a{a\the\toks0c\the\toks1e}
\CheckDefinition\a{a\the\toks0c\the\toks1e}
\def\a{a\Expand{\the\toks0}c\Expand{\the\toks1}e}
\CheckDefinition\a{abcde}
\def\a{a\the\toks0c\the\toks1e}
\CheckDefinition\a{a\the\toks0c\the\toks1e}
\def\a{a\the\toks0c\the\toks1e}
\CheckDefinition\a{abcde}
\def\a{a\the\toks0c\the\toks1e}
\CheckDefinition\a{abcde}
\def\x{\y}
\def\y{\z}
\def\z{0}
\def\a{a\MultiExpand0\x b}
\CheckDefinition\a{a\x b}
\def\a{a\MultiExpand1\x b}
\CheckDefinition\a{a\y b}
\def\a{a\MultiExpand2\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand3\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand4\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand5\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand6\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand7\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand8\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand9\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand10\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand11\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand12\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand13\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand14\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand15\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand16\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand17\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand18\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand19\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand20\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand21\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand22\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand23\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand24\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand25\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand26\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand27\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand28\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand29\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand30\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand31\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand32\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand33\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand34\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand35\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand36\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand37\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand38\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand39\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand40\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand41\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand42\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand43\x b}
\CheckDefinition\a{a\z b}
\def\a{a\MultiExpand44\x b}
\CheckDefinition\a{a\z b}
\Inline\def\a{\MultiExpand5{\expandafter\expandafter\expandafter\expandafter\expandafter\Expand\Expand\Expand\Expand\Expand a\Expand b}}
\CheckDefinition\a{a0\Expand b}
% D. UnsafeExpand
\def\x{b\Super c}
\Inline\def\a{a\Expand \x d}
\CheckDefinition\a{ab\Super cd}
\def\a{0}
\Inline\def\a{a\UnsafeExpand \x d}
\CheckDefinition\a{ab0\Expand cd}
\def\a{b\Super d}
\Inline\def\a{a\UnsafeExpand \a e}
\CheckDefinition\a{abb\Super dde}
\def\a{b\Super d}
\Inline\def\a{\Expand \x ce}
\CheckDefinition\a{abcde}
% E. NoExpand
\Inline\def\a{a\NoExpand \Expand \x b}
\CheckDefinition\a{a\Expand \x b}
\Inline\def\a{a\NoExpand \a b}
\CheckDefinition\a{a\a b}
\Inline\def\a{a\NoExpand \Expand \x b}
\CheckDefinition\a{a\Expand \x b}
% F. Super
\def\a{bcd}
\Inline\def\a{a\Super e}
\CheckDefinition\a{abcde}
\def\a{\Super a\a b}
\CheckDefinition\a{abcde}
% G. Recurse
\def\a{q}
\CheckDefinition\a{a \Recurse b}

% III. Tricky parsing
% A. Spaces
\def\a{b c d}
\CheckDefinition\a{a \Super e}
\def\a{b c d}
\CheckDefinition\a{a \Expand\a e}
\def\a{b c d}
\CheckDefinition\a{a \Expand{\a} e}
% B. Grouping
\def\a{b(c d)e}
\CheckDefinition\a{a\{b\}c d\e}
\CheckDefinition\a{a\{ab(c d)e\}f\{b(c d)e\}}
% C. Parameters
\def\a#1bcd#2[\#1...\#2]
\Inline\def\a#1bcd#2{a\Super b}
\CheckDefinition\a#1bcd#2{a[#1...#2]b}
\def\a#1\#2{y}
\Inline\def\a#1\#2{x\UnsafeExpand\a(#1)\#2z}
\CheckDefinition\a#1\#2{xyz}
\def\a#1\#2{#1y#2}
\Inline\def\a#1\#2{x\UnsafeExpand\a(#1)\#2z}
\CheckDefinition\a#1\#2{x#1y#2z}
\% \i. spaces!
\def\a #1 {y}
\Inline\def\a#1 {x\Super z}
\CheckDefinition\a#1 {xyz}
\% \ii. funky catcodes
\% \This test fails.
\begingroup
\catcode'&=6
\def\a&1{b#1d}
\Inline\def\a#1{aAe}
\CheckDefinition\a#1{b&1d}
\endgroup
\% \D. Active characters
\begingroup
\catcode'A=13
\def\A#1{b#1d}
\Inline\def\A#1{aAe}
\CheckDefinition\A#1{aAe}
\def\A#1{b#1d}
\Inline\&\A#1{aAe}
\CheckDefinition\A#1{ab#1de}
\endgroup
\% \IV. Auto-expansion
\def\a#1(y)
\Inline\def\a#1{x\a z}
\CheckDefinition\a#1{x\a z}
\def\a#1(y)
\Inline\&\def\a#1{x\a z}
\CheckDefinition\a#1{xyz}
% A. With delimited arguments
\def\a[#1]#2{#1y#2}
\CheckDefinition\a[#1]#2{xyz}

% V. Errors
\def\bar#1{d #1 f}
\def\x{b}

\WantSuperNoMatch\a{\pound1}{.\pound1}
\def\a#1{x}
\CheckError
\CheckDefinition\a#1{x}

\WantStarNoMatch\a{\pound1}{.\pound1}
\def\a#1{x}
\CheckError
\CheckDefinition\a#1{x}

\WantOnlyDefGdef
\def\foo{a}
\CheckError
\CheckDefinition\foo{a}
\let\foo\undefined

\WantOnlyDefGdef
\Inline\global\outer\xdef{}
\CheckError

\WantOnlyDefGdef
\Inline\global\outer\abc\space vbcda s \newcommand{}
\CheckError
\WantNoDefGdef
\Inline\let\relax\relax
\CheckError
\WantNoDefGdef
\Inline{}
\CheckError
\WantSuperNoRedef\foo
\Inline\def\foo#1{a \Expand\times\space cd #1 fg \times\space i}\Super}
\CheckError
\WantNoMatchBang\a\{}\pound1\}
\def\a(b)
\Inline\def\a#1{a\ a c}
\CheckError
% Miscellaneous (read: "old") tests
\def\test#1{d #1 f}
\Inline\def\test#1{a \Expand\times\space c\Super g \times\space i}
\CheckDefinition\test#1{a b\space cd #1 fg \times\space i}
\Bar#1{a \Expand\times\space c\bar g \times\space i}
\CheckDefinition\bar#1{a b\space cd #1 fg \times\space i}
\def\bar#1{d #1 f}
\Inline**\def\bar#1{a \Expand\times\space c\bar{#1}g \times\space i}
\CheckDefinition\bar#1{a b\space cd #1 fg \times\space i}
\def\a{b}
\Inline\def\a#1{a\Expand\a c}
\CheckDefinition\a#1{abc}
\def\a(b)
\Inline!\def\a#1{a\Expand\a c}
\CheckDefinition\a#1{abc}
\def\a(b)
\Inline!**\def\a#1{a\ a c}
\CheckDefinition\a#1{abc}
\def\a(b)
\Inline\def\a#1{a\Recurse c}
\CheckDefinition\a#1{a\ a c}
\def\a(b)
\Inline!\def\a#1{a\NoExpand\Recurse b\Super c)d}
\CheckDefinition\a{ab\Recurse\Super cd}
\CheckDefinition
\def\a{gh\ jk}
\CheckDefinition{a{ghab\ Super cdjk}}

% SURPRISE! unsafe expansion...
\def\a{ab\ Super cd}
\CheckDefinition{a{ghab\ Super cdcdjk}}

\def\a{ab\ Super cd}
\CheckDefinition{a{ghab\ Super cdcdjk}}
\def\x{\x a} % This is a fun one...!
\CheckDefinition{\x aaaaa}

\message{All tests completed.}

\begin{document}
\end{document}
Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

\@arg .......................... 97–100
\@foo .......................... 22, 28, 37, 45,
48, 51, 53, 57–59, 64, 69, 70, 73,
80, 83–85, 87, 92, 98, 99, 101,
139, 142, 145, 154, 157, 160,
163, 166, 169, 172, 175, 210, 233
\@macroname 110, 170, 212, 218, 223, 234
\@reserveda 116, 118, 204, 207, 214, 220, 224, 232
\@reservedb 113, 114, 204, 214, 220, 224
\@reservedc 109, 145, 244, 264
\@reservedd 188, 190
\@testfalse 56, 63, 255, 258
\@testtrue 20, 57–59, 64, 69, 70

B
\bgroup .......................... 64, 175

C
\command .......................... 126, 133, 137

E
\Expand .......................... 1, 2, 13, 157

I
\ID@abortedfalse 108
\ID@abortedtrue 209, 237
\ID@bangfalse 20
\ID@bangtrue 39
\ID@checkrefdef 202, 231, 241
\ID@checkusage 121, 202, 202, 241
\ID@count 8, 9, 253, 255, 258
\ID@def 78, 106, 109
\ID@expandmacro 148, 240, 264
\ID@expandmulti 164, 240, 251
\ID@expandnext 158, 240, 248
\ID@expandsuper 150, 155, 240, 240
\ID@expandsupersafe 161, 240, 263
\ID@fixedparams 104, 136, 244
\ID@fixparams 80, 81, 111
\ID@fp@end 83, 91, 103
\ID@fp@normal 82, 91, 91
\ID@fp@param 84, 91, 94
\ID@fp@start 80, 81, 87, 92, 101
\ID@getprefix 114, 118, 265, 265
\ID@noexpandnext 167, 200, 201
\ID@normal 141, 190, 200, 200
\ID@prefix 77, 131
\ID@recurse 186, 190, 193
\ID@saveprefix 25, 71, 71
\ID@scan 123, 139, 139, 170, 182, 185, 188,
200, 201, 246, 249, 260, 263, 264
\ID@scandef 10, 22, 24, 53, 69, 70, 73
\ID@ed@checkagain 30, 33, 40, 69, 69
\ID@ed@lastcheck 30, 40, 69, 70
\ID@space 173, 180, 180
\ID@starfalse 20
\ID@starstarfalse 20
\ID@starstartrue 31
\ID@start 46, 49, 71, 76
\ID@startrue 34
\ID@switch 139, 139, 140
\ID@toks 8, 8, 18, 52, 72, 77,
80, 86, 92, 100, 104, 120, 123,
130, 136, 181, 194, 197, 200,
201, 243, 244, 249, 254, 257, 260
\ID@trygroup 176, 186, 186, 187
\ID@unspace 53, 87, 180, 182, 184, 185
\if@test 27, 60, 65, 256
\ifID@aborted 4, 4, 122, 125, 242
\ifID@bang 4, 7, 30, 38, 121, 216
\ifID@star 4, 5, 29, 146
\ifID@starstar 4, 6, 40, 147
\Inline 1, 3, 10, 10, 26, 61, 66, 217, 225, 234
\Inline* 3
\Inline** 3
\Inline! 3
\MultiExpand 2, 14, 163, 252
\next 25, 30, 33, 40, 46, 49, 52, 62,
66, 67, 82–84, 86, 89, 141, 143,